Pflanzenwächter für die Fensterbank Teil 5 – Beleuchtungsstärkenmessung

In diesem Teil der Reihe werde ich noch einmal das wichtigste aus den vorherigen Teilen zusammenfassen. Wir haben Rückblickend schon einige Änderungen und Ergänzungen unseres Pflanzenwächters vorgenommen. Beispielsweise haben wir die Anzahl der Bodenfeuchtesensoren um 6 erweitert und einen kombinierten Temperatur- und Luftfeuchtesensor hinzugefügt. Für Pflanzen sind das schon mal eine gute Basis. Es fehlt jedoch noch für unsere Pflanzenwelt ein weiterer wichtiger ja unverzichtbarer Umweltfaktor. Dies ist das Licht, im speziellen die Beleuchtungsstärke! Genmessen wird die Beleuchtungsstärke in der Einheit Lux.
Die meisten Pflanzen fühlen sich im Beleuchtungsstärkenbereich von 300-1500 Lux (je nach Pflanzentyp) am wohlsten. Dies wollen wir natürlich berücksichtigen und mit unserem Pflanzenwächter die Helligkeit in Lux jederzeit im Auge behalten. Dazu erfassen wir die Beleuchtungsstärke in der Einheit Lux und stellen diese in gewohnter Art und Weise auf dem Handydisplay als Informationswert dar.
Für diese Aufgabe eignet sich, in unserem Fall, der Beleuchtungsstärkensensor BH1750, da dieser zum einen mit den 3,3 Volt Datenpegeln des ESP32 kompatibel ist, als auch die Beleuchtungsstärke als Datenpaket über die I2C Schnittstelle direkt in Lux Einheit an unseren ESP weitergibt.
Bestellbar ist der Lichtsensor z.B. bei AZ-Delivery Shop als Modul GY-302.
Wenn euch weitere Details zu dem Modul interessieren, findet ihr z.B. hier weitere Informationen.

Der BH1750 hat einen sehr großen Messbereich und die Auflösung kann grundsätzlich per Konfigurationsparameter zwischen 0,5 Lux, 1 Lux und 4 Lux gewählt werden.
Für unser Projekt wählen wir die mittlere Auflösung von 1 Lux aus.
Mit Hinzufügen des Lichtsensors haben wir unsere benötigten Sensoren jetzt final für das Projekt zusammen

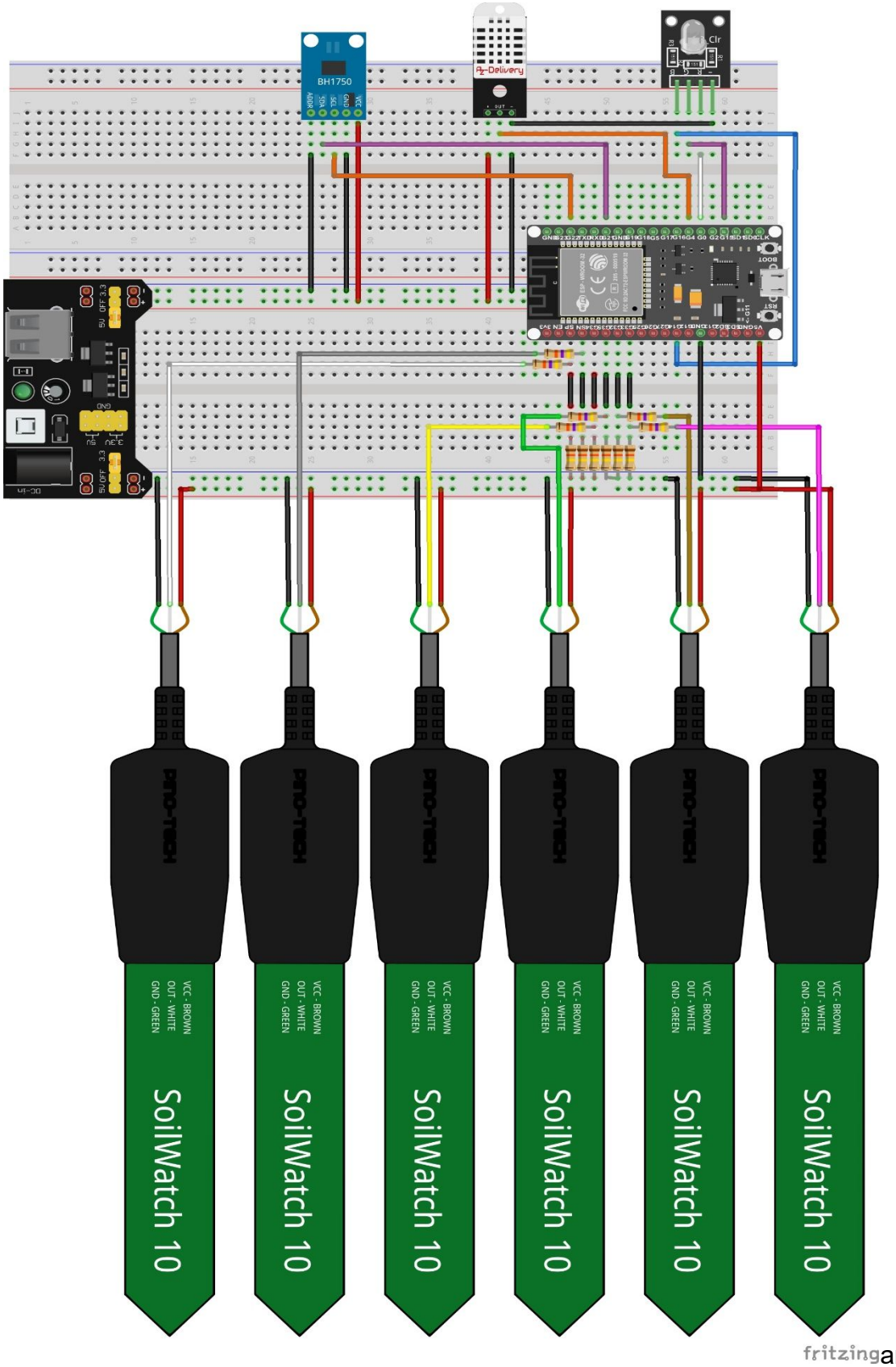
Hier findet Ihr die vorhergehenden Pflanzenwächterteile, deren Studium ich empfehlen möchte, da wichtige Informationen, Z.B. über die Kalibrierung der Bodenfeuchtesensoren darin enthalten sind:

- Teil 1
- Teil 2
- Teil 3
- Teil 4

Kommen wir zurück auf die wichtige Teileliste. Ihr findet alle relevante Teile dieses Projektes, die ihr zum Nachbau benötigt, in folgender Teileliste:

Anzahl	Beschreibung	Anmerkung
1	DHT 22	
	DHT 11	Alternativ zu DHT 22
1	KY-016 LED RGB Modul	
1	ESP-32 Dev Kit C	
6	Bodenfeuchte Sensor Modul V1.2	
1	MB102 Netzteil Adapter	Für Breadboardaufbau
1	Modul GY-302	Beleuchtungsstärkensensor
12	Wiederstände laut Beschreibung	

Wir schauen wir uns den aktualisierten Schaltplan/Verdrahtungsplan des Pflanzenwächters an:



fritzinganan:

Wir erkennen darauf den neu hinzugekommen Lichtsensor BH1570. Wir verbinden die Peripherie wie folgt:

RGB_Led Modul

RGB -Led Anode	ESP32 Pin
Rot	0
Grün	15
Blau	14

I

Bodenfeuchtesensoren

Feuchtesensor	ESP32 Pin
1	SP
2	SN
3	34
4	35
5	32
6	33

Temperatur/Luftfeuchtesensor DHT 22

PIN	ESP32 Pin
DATA /IN OUT	4

Beleuchtungsstärkensensor BH1570

BH1570 PIN	ESP32 Pin
SDA	21
SCL	22
ADDR	GND

Nachdem wir uns von der korrekten Verdrahtung überzeugt haben laden folgenden Code auf unseren ESP hoch:

```
#include <driver/adc.h>     // Build In Library. No external Library needed
//#include <esp_wifi.h>
#include <WiFi.h>
#include <WiFiClient.h>
#include <ESPmDNS.h>
#include <BlynkSimpleEsp32.h>
#include <EEPROM.h>
#include <Preferences.h>
#include <Wire.h>
#include "DHT.h"    // REQUIRES the following Arduino libraries:
                    //- DHT Sensor Library: https://github.com/adafruit/DHT-sensor-library
                    //- Adafruit Unified Sensor Lib: https://github.com/adafruit/Adafruit_Sensor


// Portedefinierung RGB LED Modul
#define LED_Rot     0     // Rote LED
#define LED_Blau    14    // Blaue LED
#define LED_Gruen   15    // Gruene LED

// I2C Port Definitionen
#define SDA1 21
#define SCL1 22

// LED PWM Einstellungen
#define PWMfreq 5000  // 5 Khz Basisfrequenz für LED Anzeige
#define PWMledChannelA  0
#define PWMledChannelB  1
#define PWMledChannelC  2
#define PWMresolution  8 // 8 Bit Resolution für LED PWM

//Zeit / Timing Definitionen für Sensorabfragen
#define ADCAttenuation ADC_ATTEN_DB_11       // ADC_ATTEN_DB_11 = 0-3,6V  Dämpfung ADC (ADC Erweiterung
#define MoisureSens_Poll_MinInterval 3600                  // Minimales Messwertübertragungsintervall zwischen zwei Bodenfeuchtemessungen in Sekunden.
#define DHT_Poll_MinInterval 2700                  // Minimales Messwertübertragungsintervall zwischen zwei Temperatur und Luftfeuchikeitsemessungen in Sekunden.
#define BH_Poll_MinInterval  1800            // Minimales Messwertübertragungsintervall zwischen zwei Lichtstärkeabfragen in Sekunden.

#define DEBUG                              // Bei definierung werden verschiedene Informationen und Messwerte auf der Seriellen Schnittstelle ausgegeben. Bitte löschen vor produktivem Einsatz !
#define MaxSensors 6                       // Maximale Anzahl an anschließbaren FeuchteSensoren
#define MinSensorValue 500                 // Mnidest AD Wert, um einen Sensoreingangskanale (1-6)  als "Aktiv" an das System zu melden. (Feuchtesensor ist angeschlossen während der Bootphase)
#define StartInit true
#define RunTime false
#define Sens_Calib true
#define Sens_NOTCalib false
```

```
#define EEPROM_SIZE 512                    // Definition Größe des Internen EEPROMS

// Blynk APP Definitionen
#define BLYNK_GREEN     "#23C48E"
#define BLYNK_BLUE      "#04C0F8"
#define BLYNK_YELLOW    "#ED9D00"
#define BLYNK_RED       "#D3435C"
#define BLYNK_BLACK     "#000000"
#define BLYNK_WHITE     "#FFFFFF"
#define BLYNK_PRINT Serial 1
#define BLYNK_NO_BUILTIN
#define BLYNK_NO_FLOAT
//#define BLYNK_DEBUG

//DHT Konfiguration
#define DHTPIN 4                        // Digital pin connected to the DHT sensor
#define DHTTYPE DHT22                   // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21                 // Falls im eingenen Projekt ein DHT 21 (AM2301) verwendet wird, bitte diee EInstellung definieren

struct SystemRunParameters
 {
int Data[MaxSensors*2] = {0,0,0,0,0,0,0,0,0,0,0,0,};  // Calibration Data für Feuchtigkeitssensor. Bitte Projekt Text beachten und Werte ensprechend anpassen
   byte StatusBorderPercentValues[MaxSensors*2][2]= { {10,50},          // Zweidimensinonales Array für Prozentgrenzwerte (Ampel) jeweils Einzeln pro Feuchtesensor (1 -6)
                                      {10,50},
                                      {10,50},
                                      {10,50},
                                      {10,50},
                                      {10,50}};
 String SensorName[MaxSensors+3] = {"Pflanze 1","Pflanze 2","Pflanze 3","Pflanze 4","Pflanze 5","Pflanze 6","Luftfeuchte","Temperatur","Lichtstärke"}; // Sensorname, der auch in der APP als Überschrift angezeigt wird
 };

struct MoistureSensorData
 {
   int Percent[MaxSensors] = {0,0,0,0,0,0};      // Feuchtigkeitssensordaten in Prozent
   byte Old_Percent[MaxSensors] = {0,0,0,0,0,0};   // Vorherige _ Feuchtigkeitssensordaten in Prozent (Zweck: DatenMenge einsparen.)
   bool DataValid [MaxSensors] = {false,false,false,false,false,false};
 };

struct DHTSensorData
 {
   float Humidity = 0 ;     // Luftfeuchtigkeitssensordaten in Prozent
   float Temperature = 0;
   float Old_Humidity = 0 ;     // Luftfeuchtigkeitssensordaten in Prozent
   float Old_Temperature = 0;
   bool DataValid  = false;
   bool SensorEnabled  = false;
 };
```

```
struct BHLightSensorData
 {
   int Lux = 0 ;          // Lichtstärke in Lux
   int Old_Lux = 0 ;      // Lichtstärke in Lux
   bool DataValid  = false;
   bool SensorEnabled  = false;
 };

DHT dht(DHTPIN, DHTTYPE); // DHT Sensor Instanz initalisieren

TwoWire I2CWire = TwoWire(0);


//Global Variables

char auth[] = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"; // Hier lt. Anleitung Auth Token dere Blynk App eintragen (E-Mail).
// Deine WiFi Zugangsdaten.
char ssid[] = "Deine_WLAN_SSID";                // Bitte an eigene WLAN SSID anpassen
char pass[] = "Dein _WLAN _Passwort!";          // Bitte an eigene WLAN Passwort anpassen
char ESPName[] = "PlantSensor1";

SystemRunParameters SysConfig;
MoistureSensorData MMeasure;
DHTSensorData  DHTMeasure;
BHLightSensorData BHMeasure;


byte AttachedMoistureSensors = 0;              // Detected Active Moisture Sensors (Count)
byte BH1750I2CAddress = 0;                     // Detected BH1750 I2C Address
bool Connected2Blynk = false;                  // Bool Variable. Stores the Connectionstate to the Blynk Cloud
unsigned long Moisure_ServiceCall_Handler = 0;  // Delay Variable for Delay between Moisure Readings
unsigned long DHT_ServiceCall_Handler = 0;      // Delay Variable for Delay between DHT Readings
unsigned long BH_ServiceCall_Handler = 0;       // Delay Variable for Delay between BH1750 Readings
unsigned long chipid;

void setup() {
  pinMode(LED_Rot,OUTPUT);
  pinMode(LED_Blau,OUTPUT);
  pinMode(LED_Gruen,OUTPUT);
  Serial.begin(115200);             // initialize serial communication at 115200 bits per second:
  I2CWire.begin(SDA1,SCL1,400000);  // join i2c bus (address optional for master)
  //chipid=ESP.getEfuseMac();         //The chip ID is essentially its MAC address(length: 6 bytes).
  //sprintf(ESPName, "%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d", chipid);
  ledcSetup(PWMledChannelA, PWMfreq, PWMresolution);
  ledcSetup(PWMledChannelB, PWMfreq, PWMresolution);
  ledcSetup(PWMledChannelC, PWMfreq, PWMresolution);
  ledcAttachPin(LED_Rot, PWMledChannelA);   // attach the channel to the GPIO to be controlled
  ledcAttachPin(LED_Blau, PWMledChannelB);
  ledcAttachPin(LED_Gruen, PWMledChannelC);
  SetLedConfig(255,255,255);
  #ifdef DEBUG
```

```
 Serial.print(F("Verbindung zu WLAN"));
 #endif
 WiFi.disconnect();
 WiFi.setHostname(ESPName);
// esp_wifi_set_mode(WIFI_MODE_STA);
 WiFi.mode(WIFI_STA);
 WiFi.begin(ssid, pass);
 WiFi.setSleep(false);
 if(WiFi.waitForConnectResult() != WL_CONNECTED)
  {
  while (WiFi.status() != WL_CONNECTED)
    {
      delay(1000);
      WiFi.begin(ssid, pass);  // try again, if first Connect is unsuccessful
      #ifdef DEBUG
      Serial.print(F("."));
      #endif
    }
  }
 #ifdef DEBUG
 Serial.println(F(" erfolgreich."));
 Serial.print(F("IP Address: "));
 Serial.println(WiFi.localIP());
 Serial.print(F("Hostname: "));
 Serial.println(ESPName);
 #endif
 if (!MDNS.begin(ESPName))
  {
   Serial.println("Error setting up MDNS responder!");
  }
 MDNS.addService("plant", "tcp", 400);

 Blynk.config(auth);   // in place of Blynk.begin(auth, ssid, pass);
 while (Blynk.connect() == false) {
     delay(500);
  Serial.print(".");
 }

 #ifdef DEBUG
 Serial.println(F("Systemkonfiguration:"));
 #endif
 if (EEPROM.begin(EEPROM_SIZE))
 {
  #ifdef DEBUG
  Serial.print(EEPROM_SIZE);
  Serial.println(F(" Bytes EEPROM"));
  #endif
 }
 Run_MoistureSensors(StartInit);
 #ifdef DEBUG
 Serial.print(AttachedMoistureSensors);
 Serial.println(F(" Bodenfeuchtigkeitsensor(en)"));
```

```
  #endif
  dht.begin();
  DHTMeasure.SensorEnabled = Run_DHTSensor (StartInit);
  if (DHTMeasure.SensorEnabled)
    {
    #ifdef DEBUG
    Serial.println(F("1 DHT 22 Sensor"));
    #endif
    }
  BHMeasure.SensorEnabled = Run_BH1750Sensor(StartInit);
  if (BHMeasure.SensorEnabled)
    {
    #ifdef DEBUG
    Serial.println(F("1 B1750 Light Sensor"));
    #endif
    }
}

void CheckConnection(){
  Connected2Blynk = Blynk.connected();
  if(!Connected2Blynk){
    Serial.println("Not connected to Blynk server");
    Blynk.connect(3333);  // timeout set to 10 seconds and then continue without Blynk
  }
  else{
    Serial.println("Connected to Blynk server");
  }
}

bool Run_BH1750Sensor (bool Init)   // Runtime Funktion für den BH170 Lichtsensor
{
byte ec;
if ((millis() - BH_ServiceCall_Handler >= BH_Poll_MinInterval *1000) | (Init))
  {
  BH_ServiceCall_Handler = millis();
  if (Init)
    {
    bool BH1750Detected = false;
    I2CWire.beginTransmission(35);
    ec=I2CWire.endTransmission(true);
    if(ec==0)
      {
      BH1750Detected = true;
      BH1750I2CAddress = 35; // BH1750 I2C Adresse ist DEC 35
      } else
      {
      I2CWire.beginTransmission(92);
      ec=I2CWire.endTransmission(true);
      if(ec==0)
        {
        BH1750Detected = true;
        BH1750I2CAddress = 92; // BH1750 I2C Adresse ist DEC 92
```

```
    }
   }
  if (BH1750Detected)
   {
   // Intialize Sensor
   I2CWire.beginTransmission(BH1750I2CAddress);
   I2CWire.write(0x01);    // Turn it on before we can reset it
   I2CWire.endTransmission();

   I2CWire.beginTransmission(BH1750I2CAddress);
   I2CWire.write(0x07);    // Reset
   I2CWire.endTransmission();

   I2CWire.beginTransmission(BH1750I2CAddress);
   I2CWire.write(0x10);    // Continuously H-Resolution Mode ( 1 lux Resolution) Weitere Modis möglich, gemäß Datenblatt
   //I2CWire.write(0x11);  // Continuously H-Resolution Mode 2 ( 0.5 lux Resolution)
   //I2CWire.write(0x20);  // One Time H-Resolution Mode  ( 1 lux Resolution)
   //I2CWire.write(0x21);  // One Time H-Resolution Mode2 ( 0.5 lux Resolution)
   I2CWire.endTransmission();

   Blynk.setProperty(V9,"color",BLYNK_WHITE);
   Blynk.setProperty(V9,"label",SysConfig.SensorName[8]);
   } else
   {
   Blynk.setProperty(V9,"label","Deaktiviert");
   Blynk.setProperty(V9,"color",BLYNK_BLACK);
   Blynk.virtualWrite(V9,0);
   return BH1750Detected;
   }
  }
I2CWire.beginTransmission(BH1750I2CAddress);
ec=I2CWire.endTransmission(true);
if(ec==0)
  {
  I2CWire.requestFrom(BH1750I2CAddress, 2);
  BHMeasure.Lux = I2CWire.read();
  BHMeasure.Lux <<= 8;                // Verschieben der unteren 8 Bits in die höhreren 8 Bits der 16 Bit breiten Zahl
  BHMeasure.Lux |= I2CWire.read();
  BHMeasure.Lux = BHMeasure.Lux / 1.2;
  BHMeasure.DataValid = true;
  if (BHMeasure.Lux != BHMeasure.Old_Lux)
   {
   BHMeasure.Old_Lux = BHMeasure.Lux;
   Update_Blynk_APP(8,true);  //  Lichtstärkeanzeige in Lux aktualisieren
   #ifdef DEBUG
   Serial.print ("Lichtstärke in Lux :");
   Serial.println (BHMeasure.Lux);
   #endif
   }
  } else
  {
  BHMeasure.DataValid = false;
```

```
    BHMeasure.SensorEnabled = false;
    Blynk.setProperty(V9,"color",BLYNK_BLUE);
    }
  }
return true;
}

bool Run_DHTSensor (bool Init)   // Runtime Funktion für den DHT Temp und Luftfeuchtesensor
{
if ((millis() - DHT_ServiceCall_Handler >= DHT_Poll_MinInterval *1000) | (Init))
  {
  DHT_ServiceCall_Handler = millis();
  DHTMeasure.Humidity = dht.readHumidity();
  DHTMeasure.Temperature = dht.readTemperature(false);   // Read temperature as Celsius (isFahrenheit = true)
  if (Init)
    {
    if (isnan(DHTMeasure.Humidity) || isnan(DHTMeasure.Temperature))
      {
      Blynk.setProperty(V7,"label","Deaktiviert");
      Blynk.setProperty(V8,"label","Deaktiviert");
      Blynk.virtualWrite(V7,0);
      Blynk.virtualWrite(V8,-20);
      Blynk.setProperty(V7,"color",BLYNK_BLACK);
      Blynk.setProperty(V8,"color",BLYNK_BLACK);
      DHTMeasure.DataValid  = false;
      return false;
      } else
      {
      Blynk.setProperty(V7,"color",BLYNK_WHITE);
      Blynk.setProperty(V7,"label",SysConfig.SensorName[6]);
      Blynk.setProperty(V8,"color",BLYNK_WHITE);
      Blynk.setProperty(V8,"label",SysConfig.SensorName[7]);
      DHTMeasure.DataValid  = true;
      DHTMeasure.Old_Humidity = DHTMeasure.Humidity;
      Update_Blynk_APP(6,true);  //  Luftfeuchteanzeige
      DHTMeasure.Old_Temperature = DHTMeasure.Temperature;
      Update_Blynk_APP(7,true);  //  Temperaturanzeige
      return true;
      }
    }
  if (isnan(DHTMeasure.Humidity) || isnan(DHTMeasure.Temperature))
      {
      Blynk.setProperty(V7,"color",BLYNK_BLUE);
      Blynk.setProperty(V8,"color",BLYNK_BLUE);
      DHTMeasure.DataValid  = false;
      DHTMeasure.SensorEnabled = false;
      return false;
      }
  DHTMeasure.DataValid  = true;
  if (DHTMeasure.Humidity != DHTMeasure.Old_Humidity)
    {
    DHTMeasure.Old_Humidity = DHTMeasure.Humidity;
```

```
    Update_Blynk_APP(6,true);  //  Luftfeuchteanzeige
    }
  if (DHTMeasure.Temperature !=  DHTMeasure.Old_Temperature)
    {
    DHTMeasure.Old_Temperature = DHTMeasure.Temperature;
    Update_Blynk_APP(7,true);  //  Temperaturanzeige
    }
  }
return true;
}

bool SetLedConfig(byte Red,byte Green,byte Blue)
{
ledcWrite(PWMledChannelA, Red); // Rote LED
ledcWrite(PWMledChannelB, Blue); // Blaue LED
ledcWrite(PWMledChannelC, Green); // Gruene LED
return true;
}

int ReadMoistureSensor_Raw_Val(byte Sensor)
  {
   int ReturnValue,i;
   long sum = 0;
   #define NUM_READS 6
   adc1_config_width(ADC_WIDTH_BIT_12);   //Range 0-4095
   switch (Sensor)
    {
    case 0:
     {
     adc1_config_channel_atten(ADC1_CHANNEL_0,ADCAttenuation);
     for (i = 0; i < NUM_READS; i++){  // Averaging algorithm
     sum += adc1_get_raw( ADC1_CHANNEL_0 ); //Read analog
     }
     ReturnValue = sum / NUM_READS;
     break;
     }
    case 1:
     {
     adc1_config_channel_atten(ADC1_CHANNEL_3,ADCAttenuation);
     for (i = 0; i < NUM_READS; i++){  // Averaging algorithm
     sum += adc1_get_raw( ADC1_CHANNEL_3 ); //Read analog
     }
     ReturnValue = sum / NUM_READS;
     break;
     }
    case 2:
     {
     adc1_config_channel_atten(ADC1_CHANNEL_6,ADCAttenuation);
     for (i = 0; i < NUM_READS; i++){  // Averaging algorithm
     sum += adc1_get_raw( ADC1_CHANNEL_6 ); //Read analog
     }
     ReturnValue = sum / NUM_READS;
```

```
    break;
    }
   case 3:
    {
    adc1_config_channel_atten(ADC1_CHANNEL_7,ADCAttenuation);
    for (i = 0; i < NUM_READS; i++){  // Averaging algorithm
    sum += adc1_get_raw( ADC1_CHANNEL_7 ); //Read analog
    }
    ReturnValue = sum / NUM_READS;
    break;
    }
   case 4:
    {
    adc1_config_channel_atten(ADC1_CHANNEL_4,ADCAttenuation);
    for (i = 0; i < NUM_READS; i++){  // Averaging algorithm
    sum += adc1_get_raw( ADC1_CHANNEL_4 ); //Read analog
    }
    ReturnValue = sum / NUM_READS;
    break;
    }
   default:
    {
    adc1_config_channel_atten(ADC1_CHANNEL_5,ADCAttenuation);
    for (i = 0; i < NUM_READS; i++){  // Averaging algorithm
    sum += adc1_get_raw( ADC1_CHANNEL_5 ); //Read analog
    }
    ReturnValue = sum / NUM_READS;
    break;
    }
   }

  return ReturnValue;
  }


void Update_Local_Display()
{
  byte red1 = 0;
  byte yellow1 = 0;
  byte green1 = 0;
  for (byte i = 0;i < AttachedMoistureSensors;i++)
    {
    if (MMeasure.DataValid[i])
       {
       if ( MMeasure.Percent[i] > SysConfig.StatusBorderPercentValues[i][1])
          {
          green1++;
          } else if ( MMeasure.Percent[i] > SysConfig.StatusBorderPercentValues[i][0])
          {
          yellow1++;
          } else
          {
```

```
          red1++;
          }
        }
    }
  if (red1 > 0)
    { SetLedConfig(255,0,0); }
  else if (yellow1 > 0)
    { SetLedConfig(255,255,0); }
  else if (green1 > 0)
    { SetLedConfig(0,255,0); }
  else
    { SetLedConfig(0,0,255); }
}

void Update_Blynk_APP(byte Sensor,bool Calibrated)
  {
    switch (Sensor)
    {
    case 0:
      {
      if ((MMeasure.DataValid[0]) & (Calibrated))
        {
        if ( MMeasure.Percent[0] > SysConfig.StatusBorderPercentValues[0][1])
          {
          Blynk.setProperty(V1,"color",BLYNK_GREEN);
          } else if ( MMeasure.Percent[0] > SysConfig.StatusBorderPercentValues[0][0])
          {
          Blynk.setProperty(V1,"color",BLYNK_YELLOW);
          } else
          {
          Blynk.setProperty(V1,"color",BLYNK_RED);
          }
        delay(100);
        Blynk.virtualWrite(V1,MMeasure.Percent[0]);
        } else
        {
        Blynk.setProperty(V1,"color",BLYNK_BLUE);
        }
      break;
      }
    case 1:
      {
      if ((MMeasure.DataValid[1]) & (Calibrated))
        {
        if ( MMeasure.Percent[1] > SysConfig.StatusBorderPercentValues[1][1])
          {
          Blynk.setProperty(V2,"color",BLYNK_GREEN);
          } else if ( MMeasure.Percent[1] > SysConfig.StatusBorderPercentValues[1][0])
          {
          Blynk.setProperty(V2,"color",BLYNK_YELLOW);
          } else
          {
```

```
        Blynk.setProperty(V2,"color",BLYNK_RED);
        }
    delay(100);
    Blynk.virtualWrite(V2,MMeasure.Percent[1]);
    } else
    {
      Blynk.setProperty(V3,"color",BLYNK_BLUE);
    }
  break;
  }
 case 2:
  {
  if ((MMeasure.DataValid[2]) & (Calibrated))
    {
    if ( MMeasure.Percent[2] > SysConfig.StatusBorderPercentValues[2][1])
        {
        Blynk.setProperty(V3,"color",BLYNK_GREEN);
        } else if ( MMeasure.Percent[2] > SysConfig.StatusBorderPercentValues[2][0])
        {
        Blynk.setProperty(V3,"color",BLYNK_YELLOW);
        } else
        {
        Blynk.setProperty(V3,"color",BLYNK_RED);
        }
    delay(100);
    Blynk.virtualWrite(V3,MMeasure.Percent[2]);
    } else
    {
      Blynk.setProperty(V3,"color",BLYNK_BLUE);
    }
  break;
  }
 case 3:
  {
  if ((MMeasure.DataValid[3]) & (Calibrated))
    {
    if ( MMeasure.Percent[3] > SysConfig.StatusBorderPercentValues[3][1])
        {
        Blynk.setProperty(V4,"color",BLYNK_GREEN);
        } else if ( MMeasure.Percent[3] > SysConfig.StatusBorderPercentValues[3][0])
        {
        Blynk.setProperty(V4,"color",BLYNK_YELLOW);
        } else
        {
        Blynk.setProperty(V4,"color",BLYNK_RED);
        }
    delay(100);
    Blynk.virtualWrite(V4,MMeasure.Percent[3]);
    } else
    {
      Blynk.setProperty(V4,"color",BLYNK_BLUE);
    }
```

```
    break;
    }
  case 4:
    {
    if ((MMeasure.DataValid[4]) & (Calibrated))
      {
      if ( MMeasure.Percent[4] > SysConfig.StatusBorderPercentValues[4][1])
         {
         Blynk.setProperty(V5,"color",BLYNK_GREEN);
         } else if ( MMeasure.Percent[4] > SysConfig.StatusBorderPercentValues[4][0])
         {
         Blynk.setProperty(V5,"color",BLYNK_YELLOW);
         } else
         {
         Blynk.setProperty(V5,"color",BLYNK_RED);
         }
      delay(100);
      Blynk.virtualWrite(V5,MMeasure.Percent[4]);
      } else
      {
        Blynk.setProperty(V5,"color",BLYNK_BLUE);
      }
    break;
    }
  case 5:
    {
    if ((MMeasure.DataValid[5]) & (Calibrated))
      {
      if ( MMeasure.Percent[5] > SysConfig.StatusBorderPercentValues[5][1])
         {
         Blynk.setProperty(V6,"color",BLYNK_GREEN);
         } else if ( MMeasure.Percent[5] > SysConfig.StatusBorderPercentValues[5][0])
         {
         Blynk.setProperty(V6,"color",BLYNK_YELLOW);
         } else
         {
         Blynk.setProperty(V6,"color",BLYNK_RED);
         }
      delay(100);
      Blynk.virtualWrite(V6,MMeasure.Percent[5]);
      } else
      {
        Blynk.setProperty(V6,"color",BLYNK_BLUE);
      }
    break;
    }
  case 6:
    {
    if (DHTMeasure.DataValid)
      {
      if (DHTMeasure.Humidity < 40) // https://www.pflanzenfreunde.com/luftfeuchtigkeit.htm und
https://www.brune.info/magazin/richtige-luftfeuchtigkeit-fuer-pflanzen/
```

```
      {
       Blynk.setProperty(V7,"color",BLYNK_RED);
       } else if (DHTMeasure.Humidity < 60)
       {
       Blynk.setProperty(V7,"color",BLYNK_YELLOW);
       } else  if (DHTMeasure.Humidity < 85)
       {
       Blynk.setProperty(V7,"color",BLYNK_GREEN);
       } else
       {
       Blynk.setProperty(V7,"color",BLYNK_YELLOW);
       }
     Blynk.virtualWrite(V7,DHTMeasure.Humidity);
     }
   break;
   }
 case 7:
   {
   if (DHTMeasure.DataValid)
     {
     if (DHTMeasure.Temperature > 43)   // https://www.spektrum.de/lexikon/biologie-kompakt/hitzeresistenz/5543
       {
       Blynk.setProperty(V8,"color",BLYNK_RED);
       } else if (DHTMeasure.Temperature < 11) // https://de.wikipedia.org/wiki/K%C3%A4ltestress_bei_Pflanzen
       {
       Blynk.setProperty(V8,"color",BLYNK_RED);
       } else
       {
       Blynk.setProperty(V8,"color",BLYNK_WHITE);
       }
     Blynk.virtualWrite(V8,DHTMeasure.Temperature);
     }
   break;
   }
 case 8:
   {
   if (BHMeasure.DataValid)
     {
     if (BHMeasure.Lux < 500)   // https://www.zimmerpflanzenlexikon.info/artikel/lichtbedarf-von-pflanzen
       {
       Blynk.setProperty(V9,"color",BLYNK_RED);
       } else if (BHMeasure.Lux < 1000)
       {
       Blynk.setProperty(V9,"color",BLYNK_GREEN);
       } else if (BHMeasure.Lux < 1500)
       {
       Blynk.setProperty(V9,"color",BLYNK_WHITE);
       } else
       {
       Blynk.setProperty(V9,"color",BLYNK_YELLOW);
       }
     Blynk.virtualWrite(V9,BHMeasure.Lux);
```

```
      }
     break;
     }
   } // End Switch
 }

void Get_Moisture_DatainPercent()
 {
 byte CalibDataOffset = 0;
 for (byte i = 0;i < AttachedMoistureSensors;i++)
  {
  CalibDataOffset =  i *2;
  int RawMoistureValue = ReadMoistureSensor_Raw_Val(i);
  if ((SysConfig.Data[CalibDataOffset] == 0) || (SysConfig.Data[CalibDataOffset+1] == 0)) // MinADC Value maxADC ADC Value
    {
    MMeasure.Percent[i] = RawMoistureValue;
    MMeasure.DataValid[i] = false;
    } else
    {
    RawMoistureValue= SysConfig.Data[CalibDataOffset+1] - RawMoistureValue;
    RawMoistureValue=SysConfig.Data[CalibDataOffset] + RawMoistureValue;
    MMeasure.Percent[i]  =  map(RawMoistureValue,SysConfig.Data[CalibDataOffset],SysConfig.Data[CalibDataOffset+1],  0,
100);
    if ((MMeasure.Percent[i] > 100 ) | (MMeasure.Percent[i] < 0 ))
     {
     MMeasure.Percent[i] = RawMoistureValue;
     MMeasure.DataValid[i] = false;
     } else  {  MMeasure.DataValid[i] = true; }
    }
  }
 return ;
 }

void Run_MoistureSensors (bool Init)   // HauptFunktion zum Betrieb der Bodenfeuchtesensoren
{
byte MinSensValue = 100;
if ((millis() - Moisure_ServiceCall_Handler >= MoisureSens_Poll_MinInterval * 1000) | (Init))
  {
  Moisure_ServiceCall_Handler = millis();
  if (Init)
   {
   for (int i = 0;i < MaxSensors;i++)
     {
     int MSensorRawValue = ReadMoistureSensor_Raw_Val(i);
     if ( MSensorRawValue > MinSensorValue)
       {
         AttachedMoistureSensors++;
       } else {break;}
     }
   if (AttachedMoistureSensors < 1)
     {
     #ifdef DEBUG
```

```
  Serial.println(F("Keine Bodenfeuchtigkeitssensoren erkannt. System angehalten."));
  #endif
  SetLedConfig(255,0,255);
  digitalWrite(LED_Rot,HIGH); // System angehalten  Led Anzeige: lila
  digitalWrite(LED_Blau,HIGH);
  digitalWrite(LED_Gruen,LOW);
  delay(1200000);
  esp_deep_sleep_start();
  while(1) {}
  }
 for (int i = 0;i < AttachedMoistureSensors;i++)
  {
  if (i == 0) {  Blynk.setProperty(V1,"label",SysConfig.SensorName[0]); }
  if (i == 1) {  Blynk.setProperty(V2,"label",SysConfig.SensorName[1]); }
  if (i == 2) {  Blynk.setProperty(V3,"label",SysConfig.SensorName[2]); }
  if (i == 3) {  Blynk.setProperty(V4,"label",SysConfig.SensorName[3]); }
  if (i == 4) {  Blynk.setProperty(V5,"label",SysConfig.SensorName[4]); }
  if (i == 5) {  Blynk.setProperty(V6,"label",SysConfig.SensorName[5]); }
  }
 for (int i = AttachedMoistureSensors;i < MaxSensors;i++)
  {
  if (i == 0) { Blynk.setProperty(V1,"label","Deaktiviert"); Blynk.setProperty(V1,"color",BLYNK_BLACK); }
  if (i == 1) { Blynk.setProperty(V2,"label","Deaktiviert"); Blynk.setProperty(V2,"color",BLYNK_BLACK); }
  if (i == 2) { Blynk.setProperty(V3,"label","Deaktiviert"); Blynk.setProperty(V3,"color",BLYNK_BLACK); }
  if (i == 3) { Blynk.setProperty(V4,"label","Deaktiviert"); Blynk.setProperty(V4,"color",BLYNK_BLACK); }
  if (i == 4) { Blynk.setProperty(V5,"label","Deaktiviert"); Blynk.setProperty(V5,"color",BLYNK_BLACK); }
  if (i == 5) { Blynk.setProperty(V6,"label","Deaktiviert"); Blynk.setProperty(V6,"color",BLYNK_BLACK); }
  }
 }
Get_Moisture_DatainPercent();
for (int i = 0;i < AttachedMoistureSensors;i++)
  {
    if (MMeasure.DataValid[i])
      {
      if (MMeasure.Percent[i] != MMeasure.Old_Percent[i])
        {
        MMeasure.Old_Percent[i] = MMeasure.Percent[i];
        if (MMeasure.Percent[i] < MinSensValue ) { MinSensValue = MMeasure.Percent[i]; };
        #ifdef DEBUG
        Serial.print(F("Feuchtigkeitswert Sensor "));
        Serial.print(i);
        Serial.print(F(" in Prozent :"));
        Serial.print(MMeasure.Percent[i]);
        Serial.println(F(" %"));
        #endif
        Update_Blynk_APP(i,Sens_Calib);     // Aktualisiere Handywerte
        }
      } else
      {
       Update_Blynk_APP(i,Sens_NOTCalib);     // Aktualisiere Handywerte
       Serial.print(F("Sensor "));
       Serial.print(i);
```

```
          Serial.print(F(" nicht kalibiert. Bitte kalibrieren. Rohdatenwert:"));
          Serial.println(MMeasure.Percent[i]);
         }
      }
   Update_Local_Display();           // Aktualisiere lokales Pflanzenwächter Display (Led)
   }

}

// Main Loop
void loop()
{
 Run_MoistureSensors(RunTime);
 if (DHTMeasure.SensorEnabled) {Run_DHTSensor(RunTime); }
 if (BHMeasure.SensorEnabled)  {Run_BH1750Sensor(RunTime); }
 Blynk.run();
}
```

In dem Code müssen vor dem ersten Hochladen **müssen** jedoch noch folgende Codezeilen an die jeweiligen eigenen Bedürfnisse angepasst werden:

```
int Data[MaxSensors*2] = {0,0,0,0,0,0,0,0,0,0,0,0,};
```

Werte bitte gemäß Beschreibung in Teil 1 der Reihe anpassen.

```
#define MoisureSens_Poll_MinInterval 3600
```

Intervall zwischen zwei Bodenfeuchtemessungen in Sekunden. Der Wert hat darauf Einfluss, wie oft die Handydaten der Bodenfeuchtesensoren aktualisiert werden, und damit auch, welches Datenvolumen für die Übertragung pro Zeit anfällt. Je höher, desto größeres Intervall. Bitte auf einen Wert setzen, der zu dem eigenen Datenvolumen bzw. Budget passt.
[Beispiel: 3600 s =1 Stunde]

```
#define DHT_Poll_MinInterval 2700
```

Intervall zwischen zwei Temperatur/Luftfeuchtemessungen in Sekunden. Der Wert hat darauf Einfluss, wie oft die Handydaten der Temperatur/Luftfeuchtemessungen aktualisiert werden, und damit auch, welches Datenvolumen für die Übertragung pro Zeit anfällt. Je höher, desto größeres Intervall. Bitte auf einen Wert setzen, der zu dem eigenen Datenvolumen bzw. Budget passt.
[Beispiel: 2700 s = 45 Minuten Datenübertragungsintervall]

```
#define BH_Poll_MinInterval  1800
```

Intervall zwischen zwei Beleuchtungsstärkenmessungen in Sekunden. Der Wert hat darauf Einfluss, wie oft die Handydaten der Beleuchtungsstärkenmessungen aktualisiert werden, und damit auch, welches Datenvolumen für die Übertragung pro Zeit anfällt. Je höher, desto größeres Intervall. Bitte auf einen Wert setzen, der zu dem eigenen Datenvolumen bzw. Budget passt. [
[Beispiel: 1800 s = 30 Minuten Datenübertragungsintervall]

```
char auth[] = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
char ssid[] = "Deine_WLAN_SSID";
char pass[] = "Dein _WLAN _Passwort!";
```

Werte bitte gemäß Beschreibung in Teil 2 der Reihe anpassen.

Es **können** folgende Parameter/Codezeilen, an die jeweiligen Bedürfnisse angepasst werden:

```
String SensorName[MaxSensors] = {"Pflanze 1","Pflanze 2","Pflanze 3","Pflanze 4","Pflanze 5","Pflanze 6"};
```

Sensornamen, der in der APP als Überschrift angezeigt wird.

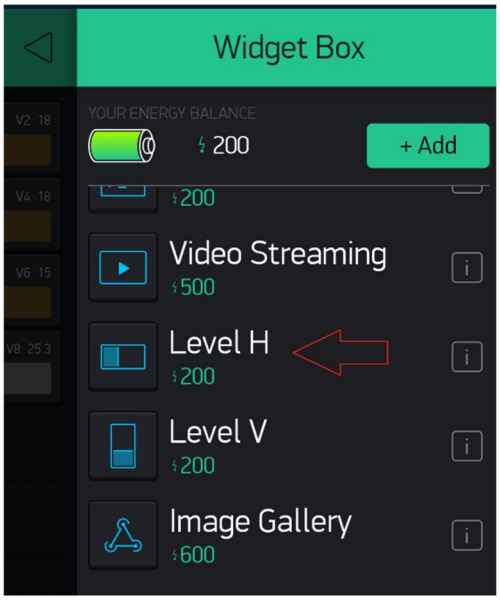
```
byte StatusBorderPercentValues[MaxSensors*2][2]= { {10,50}, …..
```

Zweidimensionales Array für Prozentgrenzwerte (Ampel) jeweils einzeln pro Feuchtesensor (1 -6). Hat Einfluss auf die „Ampel“ Anzeige und die Anzeige der Werte in der APP. Erster Wer (10) gibt die Übergangsgrenze zwischen Status „rot“ und Status „gelb“ an. Zweiter Wert gibt die Übergangsgrenze zwischen Status „gelb“ und Status „grün“ an. Beispiel: ab 51 % Bodenfeuchte „grün“ ab 9% Bodenfeuchte „rot“.

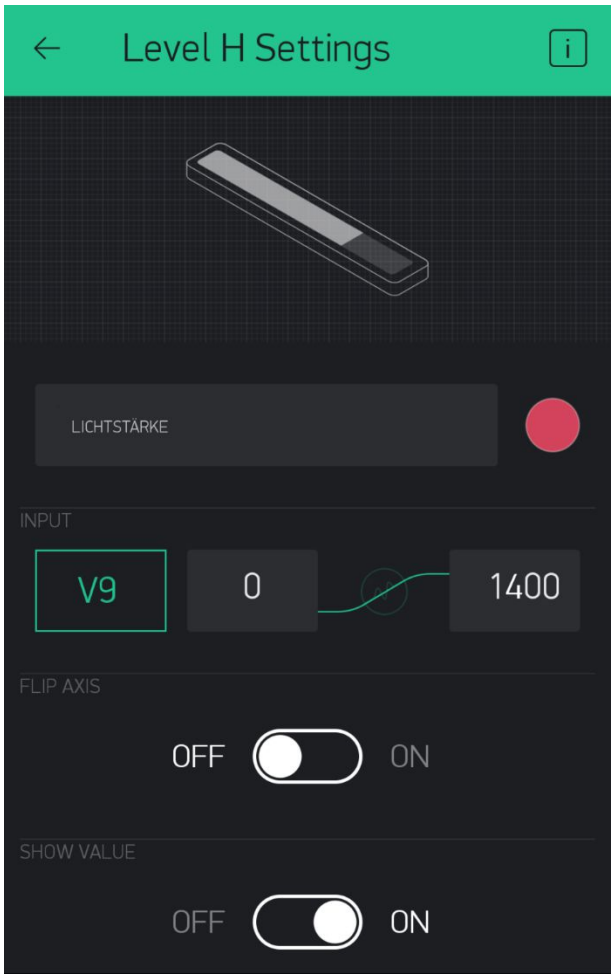
```
#define DEBUG
```

Wenn Definition „DEBUG“ vorhanden werden Laufzeitmeldungen auf der seriellen Schnittstelle ausgegeben. Für den produktiven Einsatz kanne die Definition „DEBUG“ gelöscht werden, um unnötige Ressourcenverwendung zu vermeinden..

Kommen wir nun erneut, wie in den anderen Teilen, zu der Anpassung unserer Handy APP. In dieser müssen wir nun wieder ein neues „H-Level“ Element erzeugen und dieses wie folgend beschriebene Einrichten
Wir fügen zur Anzeige der aktuellen Bodenfeuchte das „Level H“ 1-mal hinzu
.

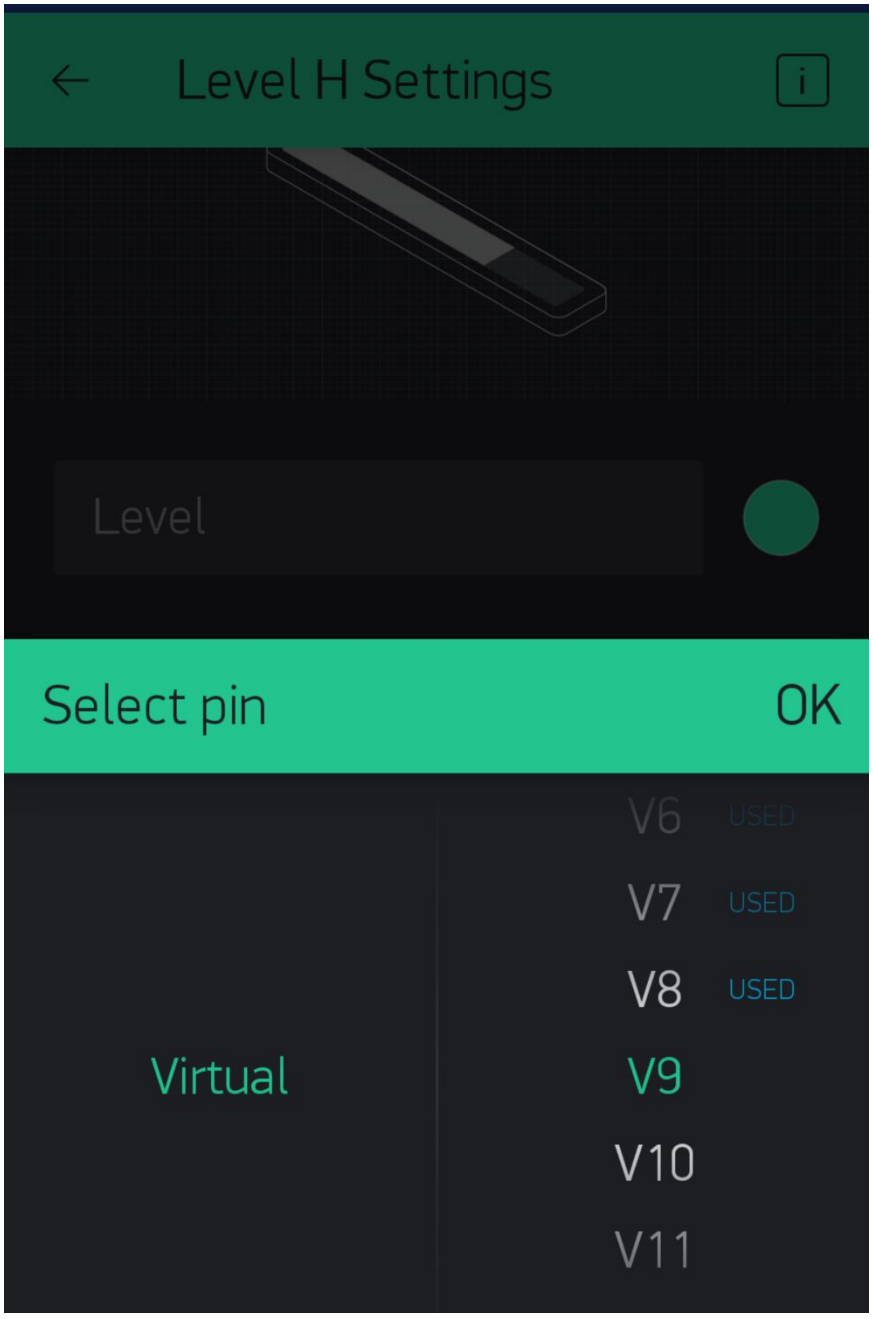


Und konfigurieren das neue Element wie folgt:

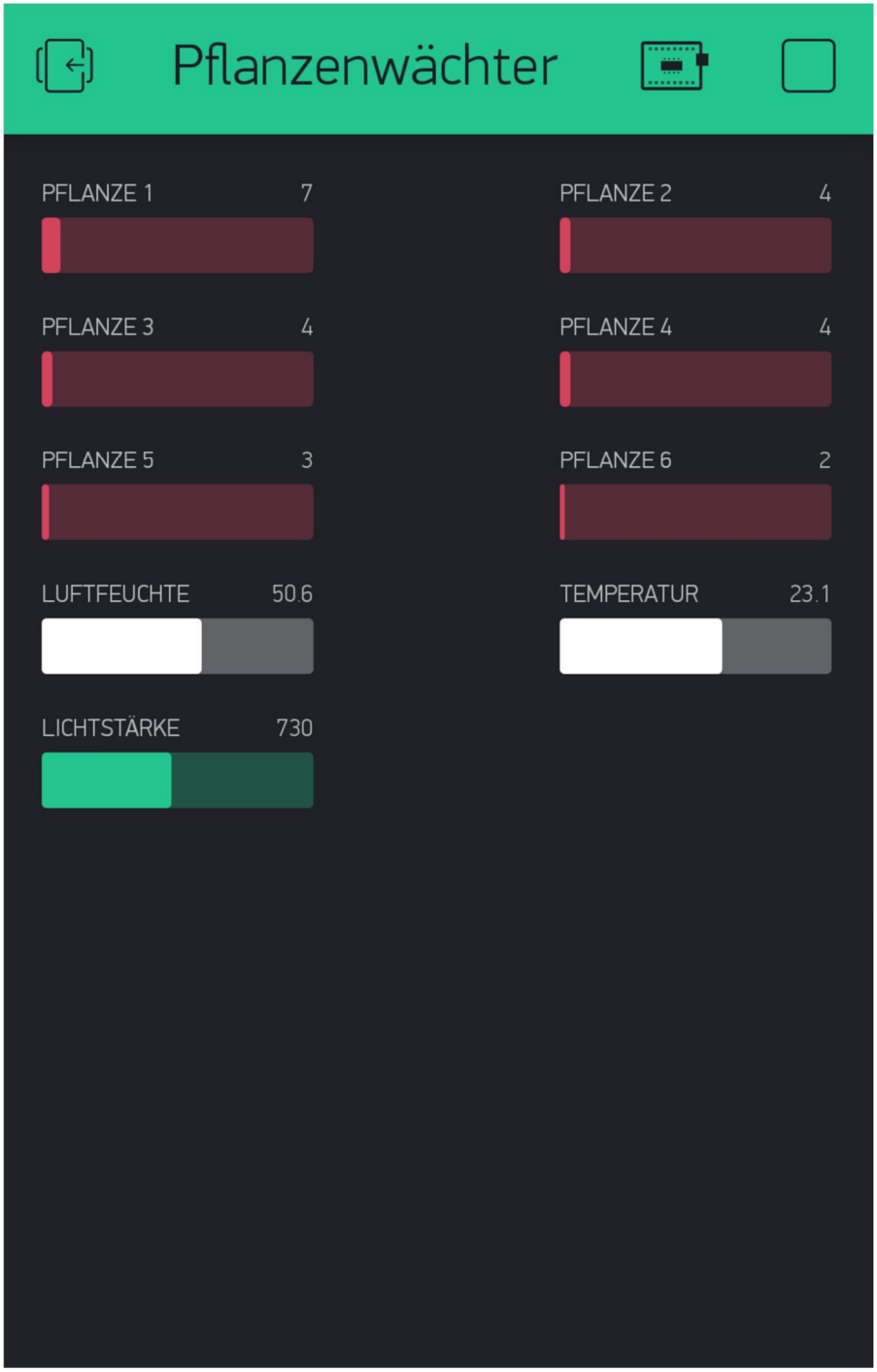


- Farbe: Rot
- Input Pin: V9 (virtuell)
- Minimum Wert: 0
- Maximum Wert: 1400 (oder 1500)

Die wichtigste Anpassung ist hierbei auch wieder die Input Variable. Diese muss fhier auf „V9“ eingestellt werden
Wir definieren wir als Reading Rate abermals „Push“



Das Endergebnis, bei Aktivierung des Projekts sollte nun ungefähr wie folgt aussehen:

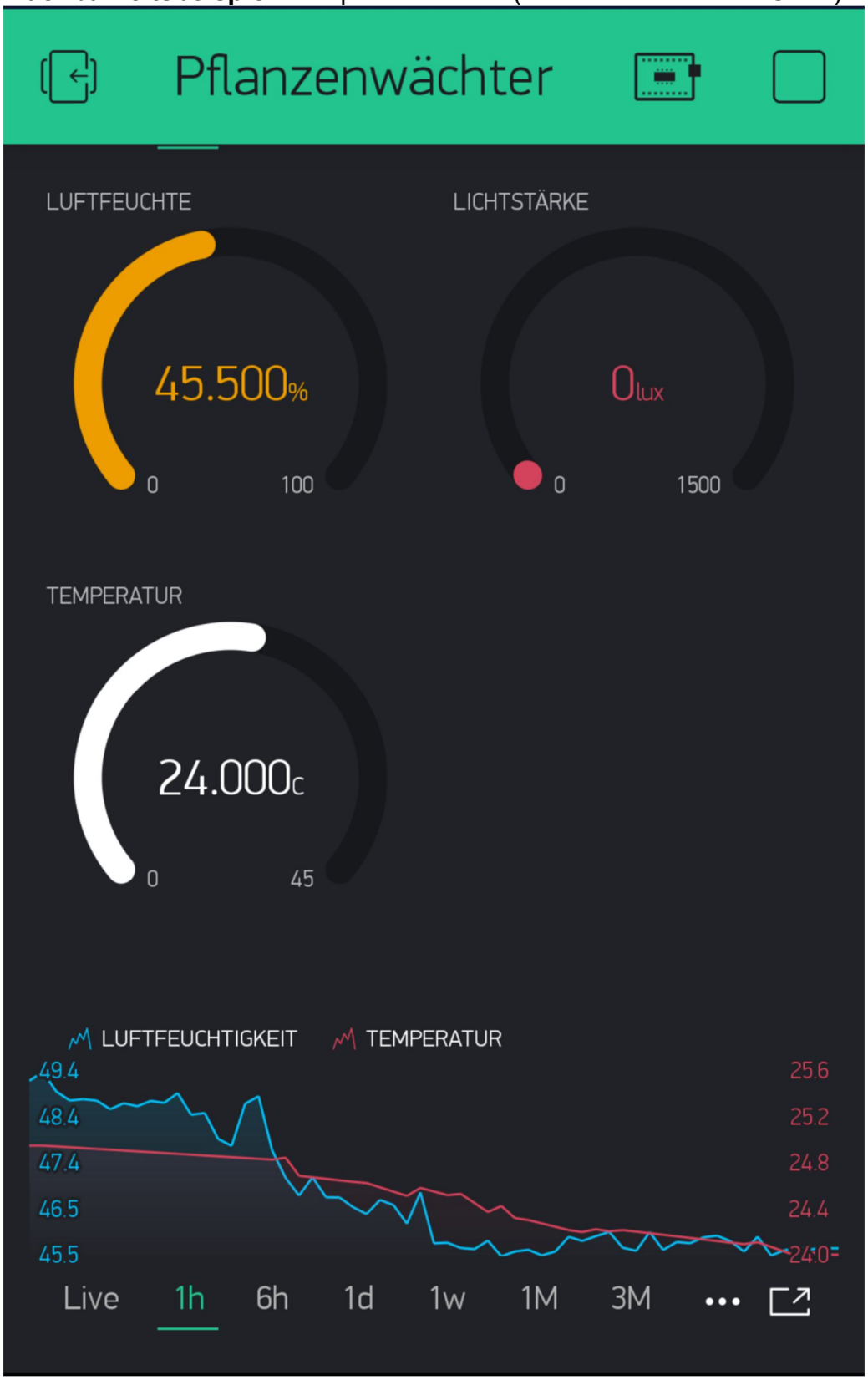


Herzlichen Glückwunsch! Du hast deinen Pflanzenwächter erfolgreich konfiguriert.
Bitte beachte folgendes:

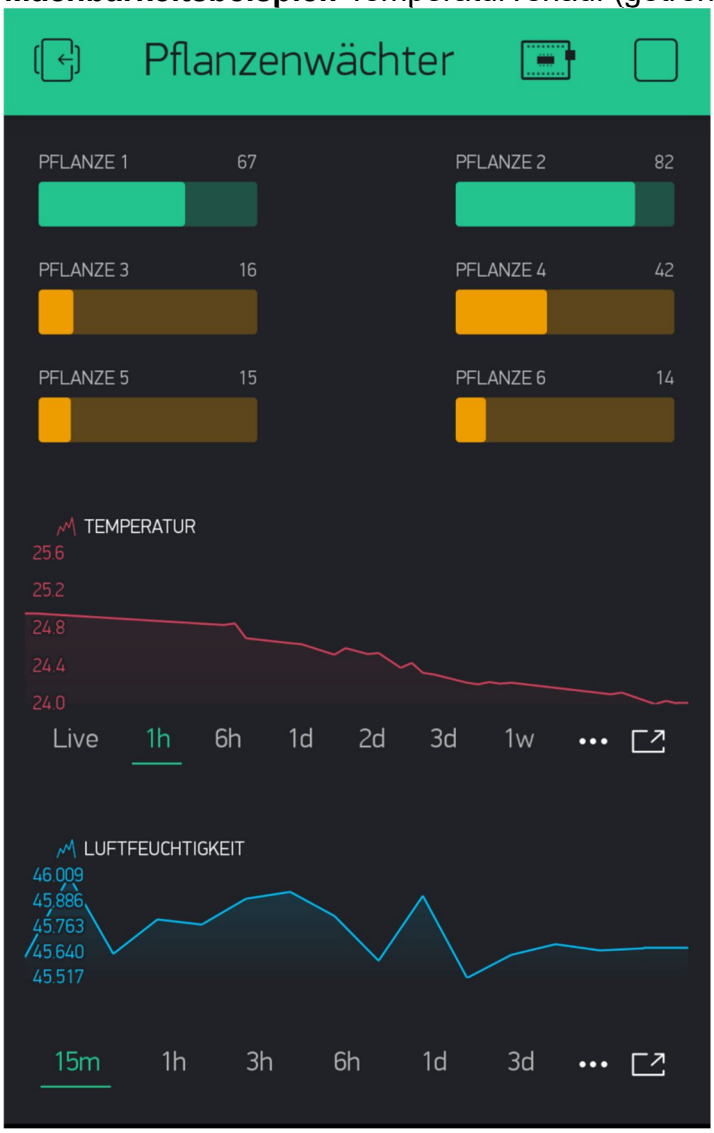
Wir haben jetzt auf unserem DashBoard 9 „H-Level“ Anzeigeelemente. Jedes „H-Level“ Element kostet uns 200 Energie. Dies macht 9*200 = 1800 Energiepunkte. Ein Hinzufügen von weiteren Elementen, die teurer als 200 Energiepunkte sind, ist ab diesem Punkt nur noch über kostenpflichtige IN-APP Käufe möglich. Die weiteren Beispiele und Screenshots mit Diagrammen sind demnach kostenpflichtig! **Zur generellen Funktion des Pflanzenwächters sind diese keinesfalls notwendig**!
Auch kann **nicht** garantiert werden, dass zukünftig die APP oder die gekauften Energiepunkte bei zukünftigen Folgereihen noch verwendet oder werden. Bitte entscheide daher eigenverantwortlich ob du Geld hierin investieren möchtest.

Auch wenn die App in jedem Fall auch OHNE weitere Investition voll Funktionsfähig und ein bereits ein echter Hingucker ist, möchte ich euch die bereits möglichen kostenpflichtigen Erweiterungen nicht vorenthalten. Dazu ist keine Änderung des ESP Codes notwendig. Alle Screenshots sind alleine über die Blynk APP unabhängig von der Firmware so konfigurierbar.

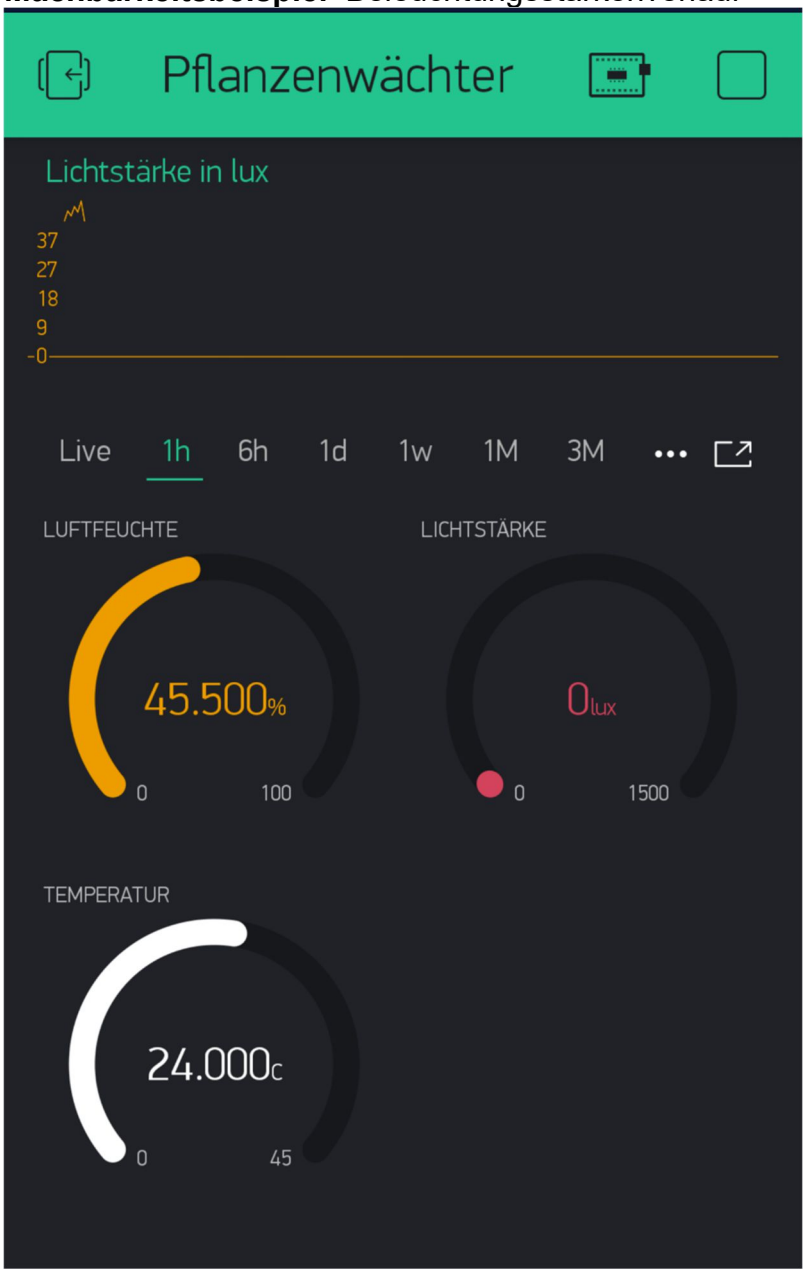
Machbarkeitsbeispiel: Temperaturverlauf (zusammen in einem Chart) :



Machbarkeitsbeispiel: Temperaturverlauf (getrennt in zwei Charts)



Machbarkeitsbeispiel Beleuchtungsstärkenverlauf



Weitere Informationen über die Blynk APP und deren Verwendung in Controllern findest du unter:

- Blynk Einführung -> https://www.blynk.cc/getting-started
- Dokumentation -> http://docs.blynk.cc/
- Sketch Generator -> https://examples.blynk.cc/
- Aktuellste Blynk Bibliothek -> https://github.com/blynkkk/blynk-library/releases/download/v0.6.1/Blynk_Release_v0.6.1.zip
- Aktuellster Blynk server -> https://github.com/blynkkk/blynk-server/releases/download/v0.41.5/server-0.41.5.jar
- Blynk Startseite -> https://www.blynk.cc

Bitte beachtet, dass dieses Projekt für Hydrokulturpflanzen oder Luftwurzlerpflanzen nicht geeignet ist. Pflanzen haben unterschiedliche Anforderungen an Ihre Umwelt. Da die individuellen Anforderungen der Pflanzen unser Pflanzenwächter nicht kennen kann, liegt die Interpretation der Wertedarstellungen des Pflanzenwächters ausschließlich in der Hand des Anwenders

Der Pflanzenwächter ist kein Ersatz für eine verantwortungsvolle Pflege der Pflanzen!

Ihr könnt dieses Projekt sowie andere Projekte von mir auch auf meiner Git-Hub Seite, finden.

Viel Spaß beim Nachbauen.